Link presentation and data transfer

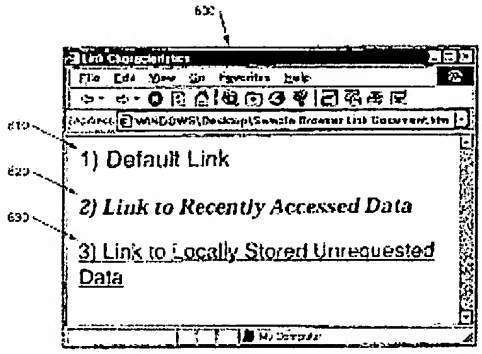
Patent number: AU2862500
Publication date: 2000-08-07
Inventor: RAZ URI; VOLK YEHUDA; MELAMED SHMUEL
Applicant: APPSTREAM INC
Classification:
- international: ***G06F17/30;*** **G06F17/30;** (IPC1-7): G06F17/30
- european: G06F17/30W9C
Application number: AU20000028625D 20000226
Priority number(s): WO2000US02190 20000126; US19990237792 19990126

Also published as:
WO0043919 (A1)

Report a data error here

Abstract not available for AU2862500
Abstract of corresponding document: **WO0043919**

A computer-implemented data access method, that may be implemented in a computer program residing on a computer-readable medium, includes receiving data at a computer from another computer prior to a user request for the data, storing the received data at the computer, and presenting a link associated with the received data using a distinguishing characteristic that differentiates the link from links to data stored only at remote computer systems. A computer system can include an interface operatively coupled to a communications link to exchange data with another computer, a display device for the presentation of data to a user, an input device to receive a user request, a memory storing executable instructions, and a processor. The processor can retrieve executable instructions from the memory to carry out aspects of the method.



Data supplied from the ***esp@cenet*** database - Worldwide

Link presentation and data transfer

Description of corresponding document: **WO0043919**

LINK PRESENTATION AND DATA TRANSFER
This is a continuation-in-part of application serial number 09/120,575 filed on July 22,1998, entitled"Streaming Modules."
BACKGROUND INFORMATION
In a client-server environment, a client computers can communicate with a server to remotely access information stored at the server. The transfer of information between the server and client computer may be provided using standard protocols and software applications. For example, a hypertext markup language (HTML) browser application at a client computer can communicate over the public Internet using TCP/IP and hypertext transfer protocols (HTTP) to receive web pages from a HTTP server. Web pages may include formatted text as well as multimedia elements, such as embedded graphics and sounds. The multimedia elements may be downloaded by the client and presented to a user by a browser application or a"plug in"browser component. Example browser applications include
Netscape Navigator 4.00 and Microsoft Internet Explorer 4.0w.

Browser applications used at client computers can use plug-in software to receive audio and video information using a streaming data transmission protocol. A streaming protocol allows information to be presented by a client computer as it is being received. For example, full-motion video can be sent from a server to a client as a linear stream of frames.

As each frame arrives at the client, it can be displayed to create a real-time full-motion video display. Audio and video streaming allows the client to present information without waiting for the entire stream to arrive at the client application. Audio and video streaming are provided by, for example, theRealAudio andRealVideo applications from
RealNetworks, Inc.

Browser applications may also make use of executable software applets to enhance the appearance of HTML-based web pages. Applets are software programs that are sent from the server to the client in response to a request from the client. In a typical applet use, HTML-based web pages include HTTP commands that cause a browser application to request an applet from a server and to begin execution of the applet. The applet may thereafter interact with a user to gather and process data, may communicate data across a network, and may display results on a computer output device. Applets may be constructed from a programming language which executes in a run-time environment provided by the browser application at the client computer. For example, theJava@ programming language from Sun Microsystems, Inc., allows Java applets to be stored at a web server and attached to web pages for execution by a Java interpreter. Java Applets, may be formed from multiple
Java Classes. Java Classes include executable Java code that can be downloaded from a server in response to a dynamically generated request to execute the class (a module execution request). If a Java Class is not available to a Java interpreter when an executing applet attempts to access functionality provided by the Class, the Java interpreter may dynamically retrieve the Class from a server. Other programming languages, such as
Microsoft Visual BasicS or Microsoft VisualC++ (E), may also be used to create applet-like software modules, such as Microsoft ActiveXTM controls.

Downloadable applets can also be used to develop large and complex programs. For example, a complex financial program may be constructed from a collection of applets. In such a financial program, separate applets may be used to gather information from a user, compute payments, compute interest, and generate printed reports. As particular program functions are required by a user, the applets associated with the required functions can be retrieved from the server. However, as the size of a software application increases, delays associated with retrieving is modules over a network likewise increase and may be unacceptable to end-users.

A user may repeatedly access a particular data item. For example, a web browser may be configured to display a particular"home page"when execution of the browser application begins. To reduce delays experienced when data such repeatedly accessed data is displayed, a browsers may store a copy in local"cache"memory following an initial access of the data.

Caching of previously accessed data may be indicated by visual highlighting in a browser display. For

example, in the Netscape Version 4.0 browser, previously accessed cached data may be indicated by altering the color of a displayed HTML link. While caching by a browser, such as is performed by Netscape Version 4.0, can improvements performance
when retrieving previously accessed data, such a caching technique does not reduce the download time when the user initially accessed the data.

SUMMARY

A browser or other computer application may be used to display links to data at remote computer, receive selections of links, and retrieve data from the remote computer in response to link selections. When data is received by a computer program from a remote computer, a copy can be stored in local memory or on a local hard disk drive in order to speed subsequent accesses to that data. When the data has been received in response to a user's request for the data, such as by a user clicking on a displayed link associated with the data, a browser history function may alter the color or other characteristic of the displayed link to indicate that the associated data has been previously retrieved and may be locally stored. Data may also be received from a remote computer and locally stored prior to a user request for the data. For example, data may be received by caching software that autonomously requests and receives data associated with links in a HTML document so that when a user request for the data is received, it can be quickly received from the cache. A user may be unaware that such autonomously retrieved data is locally stored. Advantages may be gained by increasing user awareness of locally stored data. For example, by identifying data that has been autonomously retrieved from a remote server and locally stored, a user can be made aware of data that can be quickly presented, and consequently, the user may be more likely to view such data.

In general, in one aspect, the invention features a computer-implemented data access method. The method includes receiving data at a computer from another computer prior to a user request for the data, storing the received data at the computer, and presenting a link associated with the received data using a distinguishing characteristic that differentiates the link from links to data stored only at remote computer systems.

In general, in another aspect, the invention features a computer program residing on a computer-readable medium. The program includes instructions for causing a computer to receive data from another computer prior to a user request for the data, to store the received
data, and, prior to a user request for the data, to present a link associated with the received data using a distinguishing characteristic that differentiates the link from links to data stored only at remote computer systems.

In general, in another aspect, the invention features a computer system including an interface operatively coupled to a communications link to exchange data with another computer, a display device for the presentation of data to a user, an input device to receive a user request, a memory storing executable instructions, and a processor. The processor is operatively coupled to the interface, the image display, the input device, and the memory.

The executable instructions configure the processor to receive data over the interface from another computer prior to a receipt of a request at the input device for the data, to store the received data, and, prior to a user request for the data, to present a link associated with the received data on the display device using a distinguishing characteristic that differentiates the link from links to data stored only at remote computer systems.

Implementations may include one or more of the following features. Other data may be receiving in response to a user request, stored, and presented using a different distinguishing characteristic. A link to data not stored at the computer may be presented using another different distinguishing characteristic. Presenting using distinguishing characteristics may include presenting using different fonts, colors, graphical user interface pointer shapes, and other differences. A link may be a pointer to data at another computer. A user request may include a selection of the link, and the stored received data may be presented in response to the user request. The received data may include a program module that is presented through execution of the program module, or text that is presented by displaying the text on a video display. The other computer may include a data storage storing a collection of executable modules associated with an application. The received data may include an executable module selected from the collection, and the computer may provide an execution environment for executing the application modules. The method may also include forming a module set that includes the data. The module set may be formed by selecting a sequence of modules from the collection in accordance with predetermined criteria. The sequence of modules may then be streamed to the computer from the other computer. The module sequence may be selected in accordance with a streaming control database at the other computer. The link presentation may be altered after the presentation of data associated with the link.

In general, in another aspect, the invention features a computer system having a processor operatively interconnected to a memory, a network communications device, a

graphical display device, a data storage device, and a user input device. The computer system includes a graphical user interface having interface objects that are user-selectable to present data associated with each object to a user. Each interface object may have a visual characteristics identifying the location of the objects associated data. The visual characteristics that can be associated with an interface object include a characteristic indicating that an object's associated data is stored locally at the computer system and has not been previously presented to the user and a different characteristic indicating that an object's associated data is stored locally at the computer system and has been previously presented to the user. The data stored locally at the computer system may include data received from another computer system preceding a user's selection of an interface object associated with the data from the other computer system.

The details of one or more embodiments of the invention are set forth in the accompanying drawings and the description below. Implementations may provide advantages such as facilitating access to localized data without requiring user location input. Other features, objects, and advantages of the invention will be apparent from the description and drawings, and from the claims.

DESCRIPTION OF DRAWINGS
Fig.1 illustrates a computer network.

Fig. 2 illustrates computer software application modules.

Fig. 3 is a directed graph, according to the invention.

Fig. 4 illustrates a server and a client, according to the invention.

Figs.5A-5E illustrate application code components, according to the invention.

Fig. 6 depicts a browser interface, according to the invention.

DETAILED DESCRIPTION
A computer system, such as an industry-standardIntel@ x86 compatible personal computer, an AppleMackintosh0 computer, or other computer system includes both software and hardware resources. Hardware resources may include a processor, memory, a hard disk drive, an output display device, input devices such as a keyboard and a mouse, a network communications device such as a modem or local area network interface. Software resources may include an operating system providing a graphical user interface and application programs. Such systems can function as client and server computers in a networked configuration.

Referring to Fig.1, a wide area network 100 is shown. In the network 100, a client computer101 can communicate with a server computer 102 by sending data over links 103 and 104 to a data network 130. The data network 130 may include multiple nodes 131-134 that can route data between the client101 and the server 102. The client computer101 may transmit and receive data using the TCP/IP, HTTP, and other protocols. For example, the client101 may use the HTTP protocol to request web pages from the server 102.

Web pages and multimedia data sent from the server 102 to the client 101 may have a natural linear sequence associated with them. The natural sequence of video data may be the linear order of video frames while the natural sequence of text may be the order in which pages of text are arranged in a document. Data having a natural linear sequence can be streamed from a server to a client to minimize download delays. In a streaming system, while earlier items in a liner sequence are being processed and/or displayed, subsequent items may be downloaded to the client computer. When processing and/or display of an item is complete, processing or display or a fully received"streamed"item may quickly begin. Since receipt of a streamed item is fully or partially complete when the item is requested, a user or client application requesting the streamed item will perceive a reduced downloading delay.

For example, if the first page of a document is retrieved by a user, the second page can be downloaded while the first page is being read. If the user continues reading at the second page of the document, that page will then be available at the client, such as in a cache area on a hard disk drive, and can be read without additional downloading delay.

Software execution may not follow a predictable natural linear order. Software may include jump statements, break statements, procedure calls, and other programming constructs that cause abrupt transfers of execution among sections of executing code. The execution path that is traversed during the processing of interrelated code modules (such as code segments, code classes, applets, procedures, and code libraries), will often be nonlinear, user dependent, may change with each execution of the application program, and

may change depending on the state of various data items. Although a natural order may be lacking, an advantageous order may be determined in which to stream modules. The order may be determined using criteria that is independent of the computer's internal architecture or internal operating system (execution environment) considerations.

Referring to Fig. 2, a software application 200 may include multiple modules"A" through"H."Modules"A"through"H"may be Java Classes, C++ procedure libraries, or other code modules that can be stored at a server. Some of the modules"A"through"H"may also be stored at the client computer, such as in a hard disk drive cache or as part of a software library stored at the client computer. When a client computer begins execution of the application 200, a first module, such as module"A,"may be downloaded from the server and its execution at the client410 may begin. As module"A"is being processed, the programming statements contained therein may branch to, for example, module"E."If Module"E"is not already resident at the client, the execution of module"A"can be suspended, module"E"can be retrieved from the server, and then the execution of module "E"code may begin. In such a scenario, a user will experience a module download delay associated with retrieving module"E"from the server.

To minimize module download delays experienced by a user, module"E"may be transparently streamed from a server to the client computer. Transparent streaming allows future module use to be predicted and modules to be downloaded while other interrelated modules"A"are executing. Referring to Fig. 4, an exemplary software architecture 400 providing transparent streaming is shown. The software architecture 400 includes a server 401 having a database 403 of stored software modules. The server 401 can transparently transmit a stream of software modules 405 over a communications link to a client computer 410. The communication link may be an analog modem connection, a digital subscriber line connection, a local area network connection, or any other type of data connection between the server 401 and client 410. As particular software modules are being executed at the client410, additional modules are sent from the server 401 to the client 410. In a dynamic streaming implementation, the order in which modules are streamed between the server and client may be altered based on the particular client computer 410 being served, based on the user of the client computer, and based on other dynamically determined factors.

Referring to Fig. 3, the execution order of application modules"A"through"H"may resemble a directed graph 300 rather than a linear sequence of modules. For example, as illustrated by the graph 300, after module"A"is executed, execution can continue at module"B,""D,"or"E."After module"B"is executed, execution can continue at module"C"or "G."The execution path may subsequently flow to additional modules and may return to earlier executed modules.

The server 401 can use streaming control information 402 to determine the order in which to stream modules from the server 401 to the client 410. The streaming control information 402 can include, for example, a predicted execution flow between software modules such as that represented by the directed graph 300. As downloaded modules are executed by the client 410, the client may send control data 415 to the server 401 to dynamically update and alter the order in which modules are streamed from the server 401 to the client 410. Control data 415 may be used to request particular modules from the server 401, to send data regarding the current execution state of the application program, to detail the current inventory of modules residing in the client's local storage 411, and to report user input selections, program execution statistics, and other data derived regarding the client computer 410 and its executing software.

The sequence of modules sent in the stream 405 from the server 401 to the client410 can be determined using a streaming control file 402. The streaming control file 402 includes data used by the server to predict modules that will be needed at the client 410. In a graphbased implementation, the control file 402 may represent modules as nodes of a directed graph. The control file 402 may also represent possible execution transitions between the modules as vertices ("edges") interconnecting the nodes. Referring to Table1, in a weighted graph implementation, the streaming control file 402 may include a list of vertices represent possible transitions between modules. For example, Table 1 list vertices representing all possible transitions between the modules"A"through"H"of graph 300 (Fig. 3). Each vertex in Table 1 includes a weight value indicating the relative likelihood that the particular transitions between modules will occur. In the example of Table 1, higher weight values indicate less likely transitions. The server 401 may apply a shortest-path graph traversal algorithm (also known as a"least cost"algorithm) to determine a desirable module streaming sequence based on the currently executing module. Example shortest-path algorithms may be found in TelecommunicationsNetworks: Protocols, Modeling and Analysis, Mischa Schwartz,Addison Wesley, 1987, 6.

Table 1: Graph Edge Table
EMI8.1

EdseWeight
<tb> (A,B) <SEP> 1
<tb> (A, <SEP> D) <SEP> 7 <SEP> 1
<tb> (A, <SEP> E) <SEP> 3
<tb> (B,C) <SEP> 1
<tb> (B, <SEP> G) <SEP> 3
<tb> (C, <SEP> E) <SEP> 2
<tb> (C, <SEP> G) <SEP> 6
<tb>
EMI9.1

<tb> (D, <SEP> F) <SEP> 2
<tb> (E, <SEP> H) <SEP> 2
<tb> (F,H) <SEP> 1
<tb> (G, <SEP> E) <SEP> 3
<tb> (G, <SEP> H) <SEP> 4
<tb>
For example, Table 2 represents the minimum path weight between module"A"and the remaining modules of Table1.

Table 2: Shortest Paths from ApplicationModule"A"
EMI9.2

<tb> <SEP> From <SEP> To <SEP> Shortest <SEP> Path
<tb> <SEP> Path
<tb> <SEP> Weight
<tb> <SEP> B <SEP> 1 <SEP> A-B
<tb> <SEP> C <SEP> 2 <SEP> A-B-C
<tb> <SEP> D <SEP> 7 <SEP> A-D
<tb> <SEP> A <SEP> E <SEP> 3 <SEP> A-E
<tb> <SEP> F <SEP> 9 <SEP> A-D-F
<tb> <SEP> G <SEP> 4 <SEP> A-B-G
<tb> H <SEP> 5 <SEP> A-E-H
<tb>
Based on the weight values shown in Table 2, the server 401 may determine that, during the execution of module"A", the module streamingsequence"B,""C,""E,""G,""H,""D,""F" is advantageous. If a particular module in a determined sequence is already present at the client 402, as may have been reported by control data 415, the server 401 may eliminate that module from the stream of modules 405. If, during the transmission of the sequence"B,""C,""E,""G,""H,""D,""F,"execution of module"A"completes and execution of another module begins, the server may interrupt the delivery of thesequence"B,""C,""E,""G," "H,""D,""F,"calculate a new sequence based on the now executing module, and resume streaming based on the newly calculated streaming sequence. For example, if execution transitions to module"B"from module"A,"control data 415 may be sent from the client410 to the server 401 indicating that module"B"is the currently executing module. If module "B"is not already available at the client410, the server 401 will complete delivery of module "B"to the client and determine a new module streaming sequence. By applying a shortestpath routing algorithm to the edges of Table 1 based on module"B"as the starting point, the minimum path weights between module"B"and other modules of the graph 300 (Fig. 3) can be determined, as shown in Table 3.

Table 3: Shortest Paths from Module"B"
EMI10.1

<tb> From <SEP> To <SEP> Shortest <SEP> Path
<tb> <SEP> Path
<tb> <SEP> Weisht
<tb> <SEP> C <SEP> 1 <SEP> B-C
<tb> <SEP> E <SEP> 5 <SEP> B-C-E
<tb> <SEP> B
<tb> <SEP> G <SEP> 3 <SEP> B-G
<tb> <SEP> H <SEP> 7 <SEP> B-C-E-H
<tb>
Based on the shortest path weights shown in Table 3, the server 401 may determine that module streaming

sequence"C,""G,""E,"and"H"is advantageous.

Other algorithms may also be used to determine a module streaming sequence. For example, a weighted graph 300 may be used wherein heavier weighted edges indicate a preferred path among modules represented in the graph. In Table 4, higher assigned weight values indicate preferred transitions between modules. For example, edges (A, B), (A, D), and (A, E) are three possible transitions from module A. Since edge (A, B) has a higher weight value then edges (A, D) and (A, E) it is favored and therefore, given module"A"as a starting point, streaming of module"B"before modules"D"or"E"may be preferred. Edge weight values can be, for example, a historical count of the number of times that a particular module was requested by a client, the relative transmission time of the code module, or a value empirically determined by a system administrator and stored in a table 402 at the server 401.

Other edge weight calculation methods may also be used.

Table 4: Preferred Path Table
EMI10.2

Edge <SEP> Weight
<tb> (A, <SEP> B) <SEP> 100
<tb> (A, <SEP> D) <SEP> 15 <SEP> 1
<tb> (A, <SEP> E) <SEP> 35 <SEP> 1
<tb> (B, <SEP> C) <SEP> 100
<tb> (B, <SEP> G) <SEP> 35
<tb> (C, <SEP> E) <SEP> 50
<tb>
EMI11.1

<tb> (C, <SEP> G) <SEP> 20
<tb> (D, <SEP> F) <SEP> 50
<tb> (E, <SEP> H) <SEP> 50
<tb> (F, <SEP> H) <SEP> 100
<tb> (G, <SEP> E) <SEP> 35
<tb> (G, <SEP> H) <SEP> 25
<tb>
In an preferred-path (heavy weighted edge first) implementation, edges in the graph 300 having higher weight values are favored. The following exemplary algorithm may be used to determine a module streaming sequence in a preferred-path implementation:
1: Create two empty ordered sets:
i) A candidate set storing pairs (S, W) wherein"S"is a node identifier and
"W"is a weight of an edge that may be traversed to reach node"S."
ii) A stream set to store a determined stream of code modules.

2: Let Si be the starting node.

3: Append the node Si to the Stream Set and remove any pair(Si, W) from the candidate set.

4: For each nodeSj that may be reached from node Sj by an edge(Sj, Sj) having weight Wj:
{
If Sj is not a member of the stream set then add the pair (Sj, Wj) to the candidate set.

If Sj appears in more than one pair in the candidate set, remove all but the greatest-weight (Sj, W) pair from the candidate set.

}
5: If the Candidate set is not empty
Select the greatest weight pair (Sk, Wk) from the candidate set.

Let Si = Sk
Repeat at step 3
For example, as shown in Table 5, starting at node"A"and applying the foregoing

algorithm to the edges of Table 4 produces the stream set{A, B, C, E, H, G, D, F}:
Table 5: Calculation of Stream Set
EMI12.1

<tb> Iteration <SEP> {Stream <SEP> Set}/ <SEP> {Candidate <SEP> Set}
<tb> <SEP> 1 <SEP> {A}/ <SEP> { <SEP> (B, <SEP> 100) <SEP> (D, <SEP> 15) <SEP> (E, <SEP> 35)}
<tb> <SEP> 2 <SEP> {A, <SEP> B}/ <SEP> { <SEP> (D, <SEP> 15) <SEP> (E, <SEP> 35) <SEP> (C, <SEP> 100) <SEP> (G, <SEP> 35)}
<tb> <SEP> 3 <SEP> {A, <SEP> B, <SEP> C}/ <SEP> { <SEP> (D, <SEP> 15) <SEP> (E, <SEP> 35) <SEP> (G, <SEP> 35)}
<tb> <SEP> 4 <SEP> {A, <SEP> B, <SEP> C, <SEP> E}/ <SEP> { <SEP> (D, <SEP> 15) <SEP> (G, <SEP> 35) <SEP> (H, <SEP> 50)}
<tb> <SEP> 5 <SEP> {A, <SEP> B, <SEP> C, <SEP> E, <SEP> H}/ <SEP> {(D, <SEP> 15) <SEP> (G, <SEP> 35)}
<tb> <SEP> 6 <SEP> {A, <SEP> B, <SEP> C, <SEP> E, <SEP> H, <SEP> G}/ <SEP> { <SEP> (D, <SEP> 15)}
<tb> <SEP> 7 <SEP> {A, <SEP> B, <SEP> C, <SEP> E, <SEP> H, <SEP> G, <SEP> D}/ <SEP> { <SEP> (F, <SEP> 50)}
<tb> <SEP> 8 <SEP> {A, <SEP> B, <SEP> C, <SEP> E, <SEP> H, <SEP> G, <SEP> D, <SEP> F}/ <SEP> {}
<tb>
Implementations may select alternative algorithms to calculate stream sets.

Application streaming may also be used to stream subsections of an application or module. For example, subsections of compiled applications, such as applications written in
C, C++, Fortran, Pascal, or Assembly language may be streamed from a server 401 to a client 410. Referring to Fig.SA, an application 500 may include multiple code modules such as a main code module 501 and code libraries510 and 515. The main module 501 contains program code that is executed when the application is started. The code libraries 510 and 515 may contain header data 511 and 516 as well as executable procedures 512-514 and 517-519 that are directly or indirectly called from the main module 501 and other library procedures.

In a Microsoft Windows 95/Microsoft Visual C++ implementation, the main code module 501 may contain a compiled C++"main"procedure and the library modules510 and 515 may be dynamic link libraries having compiled C++ object code procedures. Header data 511 and 516 may include symbolic names used by operating system link procedures to dynamically link libraries 510 and 515 with the main module 501. Header data may also indicate the location of each procedure within the library. In a jump table implementation, a calling procedure may access library procedures 512-514,517-519 by jumping to a predetermined location in the header 511 or 516 and from there, accessing additional code and/or data resulting in a subsequent jump to the start of the procedure.

Data and procedures within an application's code modules and libraries may be many hundreds or thousands of bytes long. Prior to executing an application, a client may need to retrieve a lengthy set of modules and libraries. By reducing the size of the module and library set, the initial delay experienced prior to application execution can be reduced. In a streaming implementation of application 500, code within subsections of the application's code modules can be removed and replaced by shortened streaming"stub"procedures. The replacement of application code with streaming stub procedures may reduce module size and associated transmission delay. For example, referring to Figs. 5A and SB, the code library 510 may include a header 511 that is 4 kilobytes (Kbytes) in length and procedures 512-514 that are, respectively, 32 Kbytes, 16 Kbytes, and 8 Kbytes. Referring to Figs. 5B and 5C, to reduce the size of the library 510, procedures code 512-514 may be removed from the library
In some scenarios, removed code, such as procedure code 512-514 which, in the example given, was replaced by stubs 515-517, may be required (called by another procedure) before it is streamed from the server 401 and integrated with the module 530. In such a case, stub code 515-516 may access streaming functions in the streaming support library 535 to obtain the required procedure. To do so, the streaming support library 535 may send control data 415 to the server 401 to request the needed procedure. In response, the server 401 can halt the current module stream 405 and send the requested module. Upon receipt of the requested module, procedures in the streaming support library 535 may be used to integrate the received module with the application and to continue with the execution of the requested module. The server may thereafter determine a new module stream based on the requested module or other control data 415 that was received from the client.

Code modules may be reduced in size without the use of stub procedures. For example, referring again to Figs. 4,SA, SB, andSE, in a interrupt driven implementation, procedure code 512-514 may be removed from

a code library 510 and stored in a database 403. Header information 511 as well as data indicating the size and location of removed procedure code 512-514 may then be transmitted to a client 410. The client 410 may construct a new library 550 by appending a series of interrupt statements in place of the removed procedure code 512-514. When the application 500 is executed, the code library 550 is substituted for the library510 and execution of the program 500 may begin. As the program 500 executes, the removed procedure code 512-514 can be streamed to the client 410 and stored in a local database 411. If the application 500 attempts to execute procedure code 512-514 it may instead execute one of the interrupt statement that have replaced procedure code 512-514. The execution of the interrupt statement halts the execution of the program 500 and transfers control to a streaming executor program 415.

Executor 415 implements interface technology similar to that of a conventional runtime object code debugger thereby allowing the executor 415 to intercept and process the interrupt generated by the application 500. When the interrupt is intercepted by the executor 415, data provided to the executor 415 as part of the client execution platform (operating system) interrupt handling functionality can be used to identify the module 550 in which the interrupt was executed and the address of the interrupt code within the module. The executor 415 then determines whether procedure code 512-514 associated with the interrupt location has been received as part of the module stream 415 sent to the client. If the appropriate procedure code has been received, the executor 515 replaces the identified interrupt with the its corresponding code. For example, procedures 512-514 may be segmented into 4 Kilobyte code modules that are streamed to the client 410. When an interrupt statement is executed by the application 500, the executor 415 intercepts the interrupt, determines an appropriate 4
Kilobyte code block that includes the interrupt statement, and replaces the determined code block with a received code module. If the appropriate code module has not yet been received, an explicit request may be sent from the client 410 to the server 401 to retrieve the code module prior to its insertion in the library 550. The executor 415 may thereafter cause the application 500 to resume at the address of the encountered interrupt.

Implementations may also stream entire modules or libraries. For example, main code module 501 may be received from the server 401 and begin execution at the client 410 while codelibraries 510 and 515 are streamed from the server 401 to the client 410. Integration of streamed modules with executing modules may be provided by client 410 dynamic module linking facilities. For example, delay import loading provided by Microsoft Visual C++ 6.0 may be used to integrate streamed modules510 and515 with executing modules 501.

Dynamic linking of streamed modules may be facilitated by storing the streamed modules on a local hard disk drive or other storage location accessible by client 410 link loading facilities. In an exemplary implementation, streaming is facilitated by altering client 410 operating system link facilities such that the link facility can send control data 415 to the
server 401 to request a particular module if the module is has not already been streamed to
the client 401.

In a protected-memory computer system, direct manipulation of executing application
code and data may be restricted. In such systems, a"kernel"level processes or procedure
may be required to support integration of streamed modules with executing application. In such a case, streaming support 535 may be pre-provisioned by installing support procedures at the client410 prior to the client's request for the application 500.

Other methods of determining stream sets may be used. In a list-based implementation, the streaming control file may include predetermined list of module streaming sequences. For example, the streaming control file 402 may include a module streaming sequence list associated with a first user and a second module streaming sequence list associated with a second user. Control data 415 sent from the client410 to the server 401 may identify the current user at the client 410. Once the user has been identified to the server, the server may stream software modules in accordance with the user's associated streaming sequence list. User-based streaming data may be advantageous where a user's past behavior can be used to anticipate the order of modules to be accessed by that user.

In graph-based streaming control file implementations, the weight of edges connecting nodes may be determined statically or dynamically and may be determined based on a collection of historical usage data. For example, in a programmer-controlled implementation, a software programmer estimate the likelihood that particular transitions between nodes will occur based on the programmer's knowledge of the software code and the expected application usage patterns. Alternatively, application profiling programs may be used to gather run-time execution data recording transitions between various applets, Classes or code modules and thereby determine the likelihood that particular transitions will occur. In a client-feedback implementation, control data 415 sent from the client 410 to the server 401 during module execution is used to build a statistical database of module usage and, based on that database, determine the module streaming order.

In a client-controlled streaming implementation, streaming control data 402 may be located at the client410 and control data 415 sent from the client410 to the server 401 may be used to sequentially request a stream of modules from the server. For example, while the client computer 410 is executing a first module, a background process may send control data 415 to a server to request additional modules that can be buffered on a hard disk 411 at the client computer410. A client-controlled streaming implementation may used existing HTTP servers and HTTP protocols to send request from the client 410 to the server 401 and send software modules from the server 401 to the client 410. Furthermore, although streaming of software modules has been emphasized in the foregoing description, non-executable data, such as hypertext markup language, binary graphic files, and text, may be streamed as a collection of modules.

Implementations may include a"handshaking"procedure whereby, at the start of application execution, control data 415 is sent between the server 401 and the client 410. The handshaking data may include an inventory of application modules residing at the client and at the server. Such handshaking data allows both the client410 and server 401 to determine their respective software module inventory and to optimize the stream of software modules based on that inventory information.

In a history-dependent implementation, a server or client can store data about a series of transitions between modules and calculate a new module stream based on a history of transitions. For example, referring to Fig. 3, if the module"G"was reached by the path A-B
G, then a server or client may determine that module"E"followed by"H"is to be streamed.

On the other hand, if the module"G"was reached by the path A-B-C-G then the streaming sequence may include only the module"H."
Locally stored data and data at remote locations may be accessed at a client computer using a data browser application. For example, NetscapeNavigator@ and Microsoft InternetExplorer@ are browser applications can display visual"links"that are selectable by a user to access data. When a link is selected by a user, data associated with the link can be retrieved from a server and presented to the user. Different data types may be presented in different ways. For example, a Java applet may be presented as an executing program, encoded audio
data may be presented as sounds reproduced by the client computer, and text may be presented visually on a graphic display.

Data identified by a link to a remote computer system can be streamed from the
remote system to a client computer prior to a user's request for the data. Data may also be obtained from remote computer systems by a caching program at the client computer that automatically retrieves data associated with links in an HTML page prior to a user's request for the data. Such streamed or autonomously cached data can be stored at a client computer in local storage space. When the link to the remote source of such data is selected, the locally stored copy of the data can be presented to a user, thereby improving the access speed to the data.

Advantages can be obtained by distinctly presenting links to data that has been
streamed, autonomously cached, or otherwise locally stored at a client computer prior to a user's request for the data. For example, by distinctly presenting such links, a user may be made aware of data that has not been viewed but which may be rapidly accessed. Such an awareness may increase the likelihood that a user will access the link to obtain data that has not been previously presented. In server-controlled streaming implementations, a streaming control database at a server can give high streaming priority to data associated with preferred links, such as links to particular advertisers. Enabling the distinct presentation of such preferred links to streamed data may provide the advantage of increasing the likelihood that such links will be selected by a user.

Different distinguishing characteristics may be used to distinctly present links to data that has been streamed, cached, or otherwise locally stored at a client computer prior to a user's request for the data. For example, as shown in Fig. 6, a browser application 600 can use a normal typeface to display an ordinary link 610, a bold-faced and italicized typeface to display a link that has been previously accessed by a user 620, and an underlined typeface to display a link 630 to data locally stored at a client computer prior to a user's request for the data. A user of the browser 600 will thereby know that unseen data associated with the link 630 can be presented with a low delay.

A determination of link presentation characteristics can be made by a browser prior to displaying link information to a user. Referring to Fig. 7, in an exemplary browser application, a link history file may be examined to determine 702 whether data associated with a link has been recently viewed. If it is determined that data associated with the link has been recently viewed, a set of characteristics identifying the link as a recently viewed link 704 is associated with the link and used when displaying the link to a user 707.

Alternatively, if the link's associated data has not been recently viewed, but is stored locally at the client computer, a set of characteristics indicating that the data is rapidly accessible may be assigned to the link 705 and used when displaying the link 707. Finally, if other presentation characteristics are not assigned, a default set of characteristics 706 may be
assigned to the link and used for presentation of the link 707. A link's presentation may be modified when certain events occur at the client computer. For example, the browser may detect activity 708 such as link selection 709 by a user or a local storage change710. The local storage change710 may occur, for example, when data is streamed from a remote computer or obtained from a remote computer by autonomous caching software prior to a user's request to access the data. After a link selection 709 (and the presentation of that link's associated data 711) or after a storage change 711 activity is detected, the browser may repeat link characteristics determination and link presentation 701.

The invention may be implemented in computer hardware, firmware, software, digital electronic circuitry or in combinations of them. Apparatus of the invention may be implemented in a computer program product tangibly embodied in a machine-readable storage device for execution by a programmable processor; and method steps of the invention may be performed by a programmable processor executing a program of instructions to perform functions of the invention by operating on input data and generating output.

The invention may advantageously be implemented in one or more computer programs that are executable on a programmable system including at least one programmable processor coupled to receive data and instructions from, and to transmit data and instructions to, a data storage system, at least one input device, and at least one output device. Each computer program may be implemented in a high-level procedural or object-oriented programming language, or in assembly or machine language if desired; and in any case, the language may be a compiled or interpreted language. Suitable processors include, by way of example, both general and special purpose microprocessors. Generally, a processor will receive instructions and data from a read-only memory and/or a random access memory.

Storage devices suitable for tangibly embodying computer program instructions and data include all forms of non-volatile memory, including by way of example semiconductor memory devices, such as EPROM, EEPROM, and flash memory devices; magnetic disks such as internal hard disks and removable disks; magneto-optical disks; and CD-ROM disks.

Any of the foregoing may be supplemented by, or incorporated in, specially-designed ASICs (application-specific integrated circuits).

A number of embodiments of the present invention have been described.

Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the invention. For example, links can be distinguished by characteristics other than, or in addition to, typeface. For example, in a graphical user interface (GUI) environment, a mouse-controlled pointer may assume a distinct shape when it is in the proximity of a link to streamed data or a particular sound may be heard when the pointer is in the proximity of the link, a unique color may be used, or a particular shape may be associated with the link. Accordingly, other embodiments are within the scope of the following claims.

Data supplied from the ***esp@cenet*** database - Worldwide

Link presentation and data transfer

Claims of corresponding document: **WO0043919**

WHAT IS CLAIMED IS: 1. A computer-implemented data access method comprising:
receiving data at a computer from another computer prior to a user request for the data;
storing the received data at the computer; and
presenting, prior to a user request for the data,[jvmii a link associated with the received data using a distinguishing characteristic that differentiates the link from links associated with data not stored at thecomputer [jvM2].

2. The method of claim 1 wherein:
the other computer comprises data storage comprising a stored collection of executable modules associated with an application;
the data received at the computer from the other computer comprises a module selected from the stored collection; and
the computer provides an execution environment for executing modules of the application.

3. The method of claim 2 further comprising:
forming a module set comprising the data by selecting a sequence of modules from the collection in accordance with predetermined criteria; and
streaming the module set to the computer from the other computer.

4. The method of claim 3 wherein selecting a sequence comprises selecting in accordance with a streaming control database at the other computer.

5. The method of claim 1 further comprising:
receiving other data at the computer in response to a user request for the other data;
storing the received other data at the computer; and
presenting a link associated with the other data using a different distinguishing characteristic.

6. The method of claim 5 further comprising presenting a link to data not stored at the computer using another different distinguishing characteristic.

7. The method of claim 6 wherein the distinguishing characteristic comprises a font and the different distinguish characteristic comprises a different font.

8. The method of claim 6 wherein the distinguishing characteristic comprises a graphical user interface pointer shape and the different distinguish characteristic comprises a different graphical user interface pointer shape.

9. The method of claim 1 wherein the link comprises a pointer to the data at the other computer.

10. The method of claim 9 further comprising:
receiving a user request comprising a selection of the link; and
presenting the stored received data in response to the user request.

11. The method of claim 10 wherein the received data comprises a program module and presenting the received data comprises executing the program module.

12. The method of claim 10 where the received data comprises text and presenting the stored received data comprises displaying the text on a video display.

13. The method of claim 10 further comprising altering the presentation of the link after presenting the stored received data.

14. A computer program residing on a computer-readable medium, comprising instructions for causing a computer to:
receive data from another computer prior to a user request for the data;

store the received data; and
present, prior to a user request for the data, a link associated with the received data using a distinguishing characteristic that differentiates the link from links associated with data not stored at the computer.

15. The program of claim 14 wherein:
the other computer comprises data storage comprising a stored collection of executable modules associated with an application;
the data received at the computer from the other computer comprises a module selected from the stored collection; and
the program residing on the computer-readable medium further comprises instructions to provide an execution environment for executing modules of the application.

16. The program of claim 15 further comprising instructions for causing the computer to:
select a sequence of modules from the collection in accordance with predetermined criteria; and
send data instructing the other computer to send the sequence of modules.

17. The method of claim 16 wherein the instructions to select a sequence of modules comprise instructions to access the predetermined criteria from a streaming control database to compute the sequence.

18. The program of claim 14 further comprising instructions for causing the computer to:
receive other data in response to a user request for the other data;
store the received other data at the computer; and
present a link associated with the other data using a different distinguishing characteristic.

19. The program of claim 18 further comprising instructions for causing the computer to present a link to data not stored at the computer using another different distinguishing characteristic.

20. The program of claim 14 wherein the link comprises a pointer to the data at the other computer.

21. The program of claim 20 further comprising instructions for causing the computerto*,
receive a user request comprising a selection of the link; and
present the stored received data in response to the user request.

22. The program of claim 21 further comprising instructions for causing the computer to alter the presentation of the link after presenting the stored received data.

23. A computer system comprising:
an interface operatively coupled to a communications link to exchange data with another computer;
a display device for the presentation of data to a user;
an input device to receive a user request;
a memory storing executable instructions; and
a processor operatively coupled to the interface, the image display, the input device, and the memory, the processor being configured by the stored executable instructions to:
(a) receive data over the interface from another computer prior to a receipt of a request at the input device for the data;
(b) store the received data; and
(c) present a link on the display device prior to a user request for the data, the link being associated with the received data and presented using a distinguishing characteristic that differentiates the link from links associated with data not stored at the computer.

24. The system of claim 23 wherein the stored executable instructions further comprise instructions to configure the processor to:
receive other data in response to a user request for the other data received at the input device;
store the received other data at the computer; and
present another link on the display device, the other link being associated with the other data and presented using a different distinguishing characteristic.

25. In a computer system having a processor operatively interconnected to a memory, a network communications device, a graphical display device, a data storage device, and a user input device, a graphical user interface comprising:
a plurality of interface objects each user-selectable to present data associated with the object to a user; and
a plurality of visual characteristics associated with interface objects, the visual characteristics comprising a characteristic associated with objects having associated data stored locally at the computer system and not previously presented to the user and a different visual characteristic associated with objects having associated data stored at the computer system which has been previously presented to the user.

26. The graphical user interface of claim 25 wherein data stored locally at the computer system comprises data received from another computer system preceding a user's selection of an interface object associated with the data from the other computer system.

27. The graphical user interface of claim 25 wherein each interface object is a hypertext markup language link.

Data supplied from the ***esp@cenet*** database - Worldwide